GlassBrowser AI Engineering Architecture Overview

Generated: 2026-01-09T17:49:14.175531Z

1. System Summary

GlassBrowser AI is an Electron desktop application that combines a Chromium browser with a trading side panel, multi-agent chat, native charting, and backtest optimization. It connects to TradeLocker (REST + streaming) and optionally to a local MT5 tick bridge. The app is split into an Electron main process for OS-level and broker integration, and a React renderer process for UI and orchestration.

2. Runtime Boundaries

2.1 Electron Main Process (electron/main.cjs)

Owns window lifecycle, IPC endpoints, secrets storage, broker adapters, and persistence. Wraps all IPC responses in a standard envelope with error classification. Hosts the TradeLocker client and trade ledger (SQLite when available; JSON fallback).

- IPC error normalization and request ID support
- TradeLocker REST + stream client coordination
- Trade ledger persistence (SQLite or JSON)
- Safe storage of API keys via Electron safeStorage

2.2 Preload Bridge (electron/preload.cjs)

Exposes a window.glass bridge to the renderer. Handles IPC invoke wrappers, stream event subscriptions, OpenAI Realtime WebSocket proxy, tab captures, and file save dialogs.

2.3 Renderer (React + Vite)

Implements all UI panels and orchestration logic. Owns chat/agents, action catalog dispatch, playbook and task tree execution, chart sessions, backtesting, and research autopilot.

2.4 Optional Python Backend (backend/mt5_bridge)

FastAPI WebSocket bridge that streams MT5 ticks to the Electron app. Runs as a local service; used by the MT5 panel for live tick feed.

3. Source Tree Map (Key Directories)

- electron/: main process, broker adapters, TradeLocker client, SQLite ledger
- components/: UI panels and shared UI elements
- services/: domain logic (chart engine, backtester, optimizer, autopilot, task trees, agents)
- hooks/: renderer state hooks
- backend/: MT5 tick bridge (FastAPI)
- resources/: static assets

4. UI Panels and Roles (components/)

4.1 Primary Panels

- ChatInterface.tsx - main trading chat, team mode, playbooks, tool cards, attachments
- NativeChartInterface.tsx - internal chart view, snapshot capture, frames
- BacktesterInterface.tsx - manual backtest, optimizer, research autopilot, batch runs
- SetupsInterface.tsx - setup watchers, library entries, watch profiles
- TradeLockerInterface.tsx - broker status, positions/orders, stream status
- AutoPilotInterface.tsx - risk config, execution mode, playbooks, truth events
- PerformanceDashboard.tsx - research session dashboards and charts

4.2 Supporting Panels

- AgentMemoryInterface.tsx - searchable agent memory store
- AuditTrailInterface.tsx - audit log view
- NotesInterface.tsx - notes and notebooks
- MT5Interface.tsx - MT5 bridge status and subscriptions
- LeaderboardInterface.tsx - rankings and metrics
- ChangesInterface.tsx - release changes
- SettingsModal.tsx - keys, models, debug controls, live error log

4.3 Shell and UX

- BrowserView.tsx / BrowserChrome.tsx - embedded web browser + tabs
- SidebarFrame.tsx / WindowFrame.tsx - layout and window chrome
- CommandPalette.tsx - quick actions
- TradeTicket.tsx - structured trade display

5. Core Service Layer (services/)

5.1 Action Catalog and Task Trees

actionCatalog.ts defines typed actions with safety gates. taskTreeService.ts provides a multi-step orchestrator with retries, dependency checks, and state snapshots.

5.2 Chart Engine and Watchers

chartEngine.ts maintains OHLC series, indicators, and pattern events. setupWatcherService.ts evaluates strategy logic against bars and emits setup signals with evidence cards. regimeClassifier.ts tags volatility, trend, and session regimes.

5.3 Backtest, Optimization, and Research Autopilot

backtestEngine.ts implements strategy simulators and metrics. backtestResearchService.ts drives broker-history grid tests. optimizerLoopService.ts and researchAutopilotService.ts run two-round optimization, regime gating, robustness checks, and session tracking.

5.4 Execution Realism Model

executionModel.ts defines slippage/spread/commission modeling, session overrides, volatility-based slippage, partial fills, and news spike perturbations for backtesting and evaluation.

5.5 Agents, LLMs, and Live Audio

openaiService.ts and geminiService.ts manage text/vision chat. openaiLiveService.ts handles realtime voice sessions. agentCapabilities.ts constrains agent permissions.

5.6 Evidence, Review, and Test Harness

evidenceCard.ts builds structured trade evidence. reviewTaxonomy.ts supports post-trade review. agentTestHarnessService.ts defines scenarios, runs, and evaluation metrics.

6. Broker Integration (TradeLocker)

TradeLocker integration is implemented in electron/tradelocker.cjs. It manages auth, account selection, REST endpoints, and streaming. It persists configuration in userData (tradelocker.json) and emits stream status events to the renderer. Streaming uses Socket.IO with token refresh, re-subscribe logic, and stale detection.

- REST: access token, accNum header, account-scoped endpoints
- Streaming: socket.io + subscriptions, SyncEnd handling, health watchdog
- Market history: cached by MarketHistoryStore
- Debug logs: tradelocker-order-debug.jsonl and tradelocker-stream.log

7. Persistence and Ledger

Primary persistence is SQLite (electron/tradeLedgerSqlite.cjs). JSON fallback uses tradeLedger.cjs. Both store ledger entries, agent memories, optimizer cache, experiment notes, research sessions, playbook runs, and optimizer winners.

7.1 SQLite Tables

- ledger_entries, ledger_memories
- agent_memories
- optimizer_eval_cache, optimizer_winners
- experiment_notes
- research_sessions, research_steps
- playbook_runs
- ledger_meta

8. System Snapshot and Truth Event Structures

types.ts defines SystemStateSnapshot, TruthEventRecord, TruthProjection, and TruthReplay. These structures unify health, chat state, broker state, chart state, and task/run status into a single snapshot for agents and diagnostics.

9. IPC and Error Plumbing

electron/main.cjs wraps IPC responses in an envelope with requestId, timing, retryable flags, and normalized error codes. This standardizes error behavior across tradelocker, broker, tradeledger, secrets, openai, gemini, and glass channels.

10. Build and Release

Vite builds the renderer bundle. Electron Builder packages the app. Scripts are defined in package.json. The installer output goes to release/. Backend MT5 bridge is shipped as extraResources.

- npm run electron:dev (local dev)
- npm run electron:pack (unpacked build)
- npm run electron:dist (installer build)

11. Module Inventory (Auto-generated)

11.1 components/

- AgentConfigModal.tsx
- AgentMemoryInterface.tsx
- AgentTestHarnessPanel.tsx
- AuditTrailInterface.tsx
- AutoPilotInterface.tsx
- BacktesterInterface.tsx
- BrowserChrome.tsx
- BrowserView.tsx
- ChangesInterface.tsx
- ChatInterface.tsx
- CommandPalette.tsx
- ErrorBoundary.tsx
- LeaderboardInterface.tsx
- MT5Interface.tsx
- NativeChartInterface.tsx
- NotesInterface.tsx
- PerformanceDashboard.tsx
- SettingsModal.tsx
- SetupsInterface.tsx
- SidebarFrame.tsx
- SystemBoot.tsx
- ToastContainer.tsx
- TradeLockerInterface.tsx
- TradeTicket.tsx
- WindowFrame.tsx

11.2 services/

- actionCatalog.ts
- agentCapabilities.ts
- agentTestHarnessService.ts
- audioService.ts
- autoPilotPolicy.ts
- backtestCompute.worker.ts
- backtestComputeWorkerClient.ts
- backtestEngine.ts
- backtestResearchService.ts
- chartEngine.ts
- evidenceCard.ts
- executionModel.ts
- geminiService.ts
- notesStorage.ts
- openaiLiveService.ts
- openaiService.ts
- optimizerLoopService.ts
- regimeClassifier.ts
- researchAutopilotService.ts
- reviewTaxonomy.ts
- setupLibraryService.ts
- setupSignalLifecycle.ts
- setupWatcherService.ts
- stringHash.ts
- taskTreeService.ts
- tradeIdentity.ts
- tradeValidation.ts
- tradingView.ts
- url.ts

11.3 hooks/

- useChartSessions.ts
- useChat.ts
- useMt5Bridge.ts
- usePortfolio.ts
- useSidebar.ts
- useTabs.ts
- useTradeLocker.ts